

COMPENG 2SH4 Project – Peer Evaluation [25 Marks]

Your Team Members	arifm21	hadim4
Team Members Evaluated	danemanj	tandod2

Provide your genuine and engineeringly verifiable feedback. Ungrounded claims will lead to deductions. Completing the peer code evaluation on time will earn your team a total of **25 marks**. Do not exceed 2 paragraphs per question.

Peer Code Review: OOD Quality

1. **[3 marks]** Examine the main logic in the main program loop. Can you easily interpret how the objects interact with each other in the program logic through the code? Comment on what you have observed, both positive and negative features.
2. **[3 marks]** Quickly summarize in point form the pros and cons of the C++ OOD approach in the project versus the C procedural design approach in PPA3.

Peer Code Review: Code Quality

1. **[3 marks]** Does the code offer sufficient comments, or deploys sufficient self-documenting coding style, to help you understand the code functionality more efficiently? If any shortcoming is observed, discuss how you would improve it.
2. **[3 marks]** Does the code follow good indentation, add sensible white spaces, and deploys newline formatting for better readability? If any shortcoming is observed, discuss how you would improve it.

Peer Code Review: Quick Functional Evaluation

1. **[3 marks]** Does the Snake Game offer smooth, bug-free playing experience? Document any buggy features and use your COMPENG 2SH4 programming knowledge to propose the possible root cause and the potential debugging approaches you'd recommend the other team to deploy. (NOT a debugging report, just technical user feedback)
2. **[3 marks]** Does the Snake Game cause memory leak? If yes, provide a digest of the memory profiling report and identify the possible root cause(s) of the memory leakage.

Project Reflection

Recall the unusual objPos class design with the additional Pos struct. After reviewing the other team's implementation in addition to your own, reflect on the following questions:

1. **[3 marks]** Do you think the compound object design of objPos class is sensible? Why or why not?
2. **[4 marks]** If yes, discuss about an alternative objPos class design that you believe is relatively counterintuitive than the one in this project. If not, explain how you'd improve the object design. You are expected to facilitate the discussion with UML diagram(s).

Peer code review (2 paragraphs per question)

Project reflection (max of 2 pages)

(Work is below)

OOD QUALITY

1. The main game loop is kept very simple relying solely on function calls. This makes things very easy to keep track of given the main condition `while(!game->getExitFlagStatus())`, understanding that as long as the get exitflag status remains false, the game will continue running in the order of function calls. Additionally, outside of this status is the simple `CleanUp()` function, suggesting that once the exit status is no longer false, the board will be cleaned up.

Adding on to the neatness and organization of the main function. Nothing is hard coded within it which is excellent practice. Moreover, to this excellent practice, the main function is at the top of the `project.cpp` file and right underneath any important features such as including different libraries, header files, universal variables, calling classes, and initializing the functions beforehand. Thus, making it very easy to see the primary tools, and classes used to code the project. If I were to be picky, comments regarding some of these values may be beneficial especially if there are several similar classes (which is not the case in the scope of this project). For example, adding a comment after the following line: `Player* player;` such as `//in charge of player movement, wraparound logic, and food collision;` may make it easier to debug certain aspects of the game especially if this were a larger project.

2. Pros:
 - Modular design for all classes (Player, Food, GameMechs), allows for reuse in other projects, and easier to maintain if changes had to be made to the code. For example, if you wanted to change the food generation pattern, type, etc. It can be done solely through the Food class.
 - Additionally, the usage of encapsulation prevents extra bugs. Comparing to C, you can't have encapsulation meaning a struct, for example could be overwritten or the cause for errors/bugs. But in this group's code they utilized encapsulation, for example they made the `foodList`, a private method since it should not be accessed whereas they made other public methods accessible.
 - Scalability is another important feature when it comes to using C++ OOD approach, as additional classes, and more functions can be implemented without interfering with the already existing code. Not as present in this project as it is a relatively smaller project, but this team clearly showed that they can add an entire feature using an OOD approach as they created an entire new food class. This can be continued for other features like levels, barricades, enemies, etc. Overall, the OOD approach makes it easy to add on to code unlike C since you'd have to make much more changes in several areas that may not be as easily reversible in case of error like you can in C++

Cons:

- C++ OOD makes debugging significantly harder than a straightforward C approach. With C++ OOD, multiple methods in different classes are all interacting at once so it may be harder to find the issue at hand since there is a lot more to keep track of. Whereas in C, since it is all procedural you can simply follow along without having to jump from class to class, at most you'd be jumping from function to function, which makes finding errors and bugs a lot simpler than an OOD approach.
- Memory handling is another risk when it comes to a project like this one. You can see in this project the group did a good job with 0 bytes of memory leak, however the approach of manually creating and

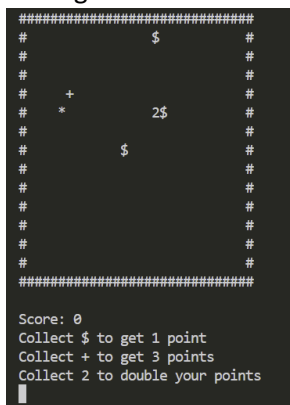
removing memory allocations (with new and delete) often leads to memory management issues. Luckily this project was small enough and the group did well, however there are external tools and libraries that support developers with memory management. Whereas in C, there is still external tools and methods to help developers manage their memory but there is significantly more, and better ones for C++ where you can take an OOD approach.

CODE QUALITY

1. Their code offers sufficient comments, they generally explain every function within their game and why they did what. The coding style itself is presented in a very neat fashion. Essentially, if someone who understands basic syntaxes is new to programming, or someone in their first week of 2SH4, they would be able to understand what everything in the code is doing and why they did what they did.
2. Yes. After reading their code, I'd say they do, the code follows good indentation and has sensible white spaces. Nothing within their code is spaced unevenly or done without a purpose. Moreover, they implemented newline within their statements for an efficient user experience. Moreover, with their overall coding efficiency, neatness, and effectiveness I'd say there is nothing wrong, as I, myself would implement my structure the exact same way, if I was to do this project again

QUICK FUNCTIONAL EVALUATION

1. Yes, the snake game is smooth and runs with no bugs. The point system, wrap around, movement, and loss (collision with body), and the rest of the game all work flawlessly. However, there is no option for the player to quit the game in the middle of playing that is shown when playing the game. Rummaging through the code I found out that you can click ESC to exit the game (both in the middle of the game and after losing), however it is not shown on the terminal when playing so the only way for a player to know would be to look at the code. I have included a screenshot of the terminal once you start to play despite no bugs it is not clear that ESC needs to be clicked to exit the game early.



```
#####
#           $           #
#           #           #
#           #           #
#   +       #           #
#   *       2$        #
#           $           #
#           #           #
#           #           #
#           #           #
#           #           #
#           #           #
#           #           #
#####

Score: 0
Collect $ to get 1 point
Collect + to get 3 points
Collect 2 to double your points
█
```

2. No there is no memory leaks present after running Dr.Memory.

PROJECT REFLECTION

1. The objPos is responsible for updating and tracking the objects around the board, so it is important. However, yes, I believe objPos was somewhat sensible. Firstly, it had very good reusability within their code, along with our code as well, as it was implemented within many parts of the code. The code itself within the objPos class is done in an effective manner in my opinion as it helped a lot within the code. Hypothetically, if we were to expand the game where we had to add more complex features, the current objPos class is written in a way where it would be very easy to work around it and simply add features without changing or rewriting the entire code.
2. The objPos is sensible as another alternative is to simply 'brute force' your way through it. This class is an encapsulation of the Pos struct. Allowing for reusable code as mentioned in the previous question. The alternative could be to have the position of *every* item (food, snake + body, or if you were to update the game any other items like barricades, other food like items, etc.). Having the raw position data of each item makes things significantly harder to maintain and update. You would need separate logic for every item, making you, for example, check boundaries for each item instead of checking them once and having everything spawn inside of these boundaries. Overall, the objPos does make sense in this project, but is not needed as this project is relatively smaller, with no further updates or maintenance needed.